UNILAB NETWORK

DISPENSA DI

COMPUTER SCIENCE: PYTHON

EDIZIONE A.A. 2022 - 2023

A cura di Domenico Destito e Chiara Tua

INTRODUZIONE ALLA PROGRAMMAZIONE

La programmazione consiste nell'istruire una macchina a svolgere un compito, che può essere sia semplice che complesso. Un programma, quindi, è un insieme di istruzioni che un computer segue per completare un'attività specifica. Programmare permette di istruire un computer a raccogliere grandi quantità di dati ed elaborarli in maniera efficacie. Oltre ad essere utilizzate per scopi informatici tradizionali, le competenze di programmazione sono oggi richieste in molti altri campi, come il web marketing, l'analisi dei big data e molte altre tecnologie.

MEMORIZZAZIONE DATI E COME FUNZIONAMENTO DI UN PROGRAMMA

Per memorizzare dati, sono necessarie tre componenti dell'hardware: la CPU (Central Processing Unit), che svolge le operazioni di base specificate nel programma, la memoria (in forme come la ROM, la RAM e le memorie di massa) e i dispositivi di input/output (ad esempio, la tastiera). Queste componenti comunicano tra loro attraverso canali noti come BUS. Tuttavia, la CPU comprende solo il linguaggio macchina, che è costituito da sequenze di 0 e 1 (notazione binaria). Di conseguenza, è necessario utilizzare linguaggi di programmazione ad alto livello (come Python), le cui istruzioni vengono tradotte in linguaggio macchina. È possibile tradurre quindi informazioni di diverso tipo in sequenze di 0 e 1, come lettere, numeri, colori dei pixel e suoni.

Per eseguire un programma, è necessaria una traduzione del codice sorgente. Questa operazione può avvenire in due modi:

- Compilazione: il programma sviluppato viene compilato e produce un eseguibile (ad esempio, un file .exe) che può essere utilizzato direttamente. Questo metodo ha il vantaggio della velocità, ma presenta come svantaggio il fatto che il programma viene compilato tutto insieme, rendendo più difficile individuare eventuali errori (ad esempio, in C).
- Interpretazione: il programma sviluppato viene tradotto ed eseguito istruzione per istruzione. Questo metodo rende più lenta l'esecuzione, ma rende più semplice individuare eventuali errori (ad esempio, in Python)

COME SI SONO EVOLUTI I LINGUAGGI?

Agli inizi dell'informatica, la programmazione in linguaggio macchina era essenziale, ma poiché già negli anni '40 era molto complessa, negli anni '50 fu introdotto il linguaggio Assembly, considerato di basso livello, ma che richiedeva comunque il proprio "traduttore": l'Assembler. Successivamente, furono sviluppati i primi linguaggi di alto livello, come COBOL, seguito da C, Java, C++, Python e così via. Questi linguaggi utilizzano una sintassi molto simile all'inglese e possono essere suddivisi in linguaggi dichiarativi, che descrivono la realtà e dichiarano un obiettivo richiesto, e linguaggi procedurali, che definiscono algoritmi che portano a un obiettivo. Python può essere considerato un linguaggio sia procedurale che dichiarativo.

CICLO DI SVILUPPO, ALGORITMI E DIAGRAMMI DI FLUSSO

Nel processo di sviluppo di un programma, si inizia dalla definizione dei **requisiti** che il software dovrà soddisfare. Successivamente, si passa alla fase di **progettazione,** dove si definisce in modo generale cosa il programma dovrà fare. Successivamente, si procede con **l'implementazione** di tali linee guida nel vero e proprio codice del programma. Successivamente, si eseguono dei test al fine di individuare e correggere eventuali errori (questa fase è chiamata **debugging**).

Nelle fasi di progettazione e implementazione, si utilizzano gli **algoritmi**, che sono definiti come una sequenza finita di azioni codificate che portano alla soluzione di un problema. Le istruzioni di un algoritmo devono essere finite, materialmente eseguibili, non ambigue e portare a un risultato.

Gli algoritmi possono essere rappresentati attraverso **diagrammi di flusso**, utilizzando elementi geometrici con significati specifici: ovali per l'inizio e la fine dell'algoritmo, romboidi per l'input e l'output dei dati, rettangoli per le azioni elementari, rombi per gli elementi di decisione a scelta e le frecce che indicano la direzione di esecuzione.

INTRODUZIONE A PYTHON

CHE COS'È PYTHON?

Python è un linguaggio di codifica interpretato, **interattivo** e a oggetti di elevato livello. Questo idioma è stato distribuito nel 1991 e da allora sono state create numerose versioni, che hanno gradualmente aumentato la sua diffusione, grazie anche alla sua facilità di utilizzo rispetto ad altri linguaggi. Inoltre, la popolarità di Python è stata sostenuta dal suo essere un linguaggio open source, costantemente ampliato con librerie, e dalla sua compatibilità con tutti i sistemi operativi più diffusi.

Come abbiamo precedentemente indicato, Python è un linguaggio interattivo, che consente di scrivere varie istruzioni direttamente nella sua **shell.** Inoltre, è orientato agli oggetti, il che significa che la soluzione dei problemi non viene affrontata solo come sequenza di istruzioni, bensì in termini di oggetti e dei relativi attributi (per una migliore comprensione, vedi l'ultimo capitolo del documento). Infine, in Python è fondamentale la tipizzazione dinamica delle variabili: in questo linguaggio, le variabili sono uno degli elementi chiave e contengono i dati o i valori ottenuti dai calcoli e dalle operazioni eseguite dal programma.

IDLE: EDITOR E SHELL

L'**IDLE** (Integrated Development and Learning Environment) è l'ambiente di sviluppo incluso al momento dell'installazione di Python, anche se è possibile utilizzare ambienti diversi. È costituito da due componenti, la console e l'editor.

• La **console** è l'interprete dei comandi di Python, che viene utilizzato in modalità interattiva e consente di eseguire i comandi uno per volta dopo averli digitati. Questa modalità di utilizzo viene anche chiamata "linea di comando" ed è utile quando si sperimentano brevi comandi prima di scrivere il programma vero e proprio nell'editor.

o La console è il primo foglio che viene aperto una volta avviato il programma; mostra in cima un messaggio di apertura seguito dal prompt (>>>) che indica che il programma è pronto a ricevere nuove istruzioni.

o Se vengono inseriti comandi errati nella console, viene mostrato immediatamente un messaggio di errore una volta premuto invio.

o Nella console non è possibile modificare o cancellare il comando scritto, ma va riscritto. Inoltre, non è possibile ripulire la console, ma va riavviata (chiudere e riaprire oppure selezionare in alto Console -> Riavvia oppure premere Ctrl + F6).

o Infine, una caratteristica fondamentale della console è che i dati non vengono salvati una volta chiuso il programma.

• L'**editor**, che si apre selezionando *File -> Nuovo file* oppure premendo *Ctrl+N*, viene utilizzato solo in modalità script. A differenza della console, consente di salvare i file con estensione .py in modo da poterli modificare ed eseguire in seguito. Mentre nella console i comandi venivano eseguiti ogni volta che si premeva invio, nell'editor vengono scritti i programmi per intero, che poi possono essere eseguiti (selezionando in alto *Esegui -> Esegui modulo* oppure premendo il *tasto F5).* Selezionando in alto nella scheda il pulsante *Opzioni -> Configura IDLE* si accede alle diverse opzioni di IDLE, che possono essere modificate in qualsiasi momento e che contengono, tra le altre cose, anche le scorciatoie da tastiera (shortcut) che consentono di lavorare più agevolmente. Una combinazione importante è quella della storia-precedente (*ALT+p),* che richiama l'ultimo comando inserito che si vuole riscrivere. Nella scheda Evidenziazione è possibile vedere a cosa corrisponde ogni colore utilizzato durante la programmazione. Questo linguaggio, infatti, ha dei colori assegnati ai vari tipi di elemento (che vengono inseriti automaticamente mentre si scrive):

• Viola: funzioni integrate (ad esempio: stampa);
• Verde: stringhe di testo (tra apici o virgolette), ma anche i commenti inseriti fuori dalle righe di codice.
• Blu: output dell'istruzione o nomi di funzione (come funzioni definite dall'utente e classi);
• Rosso: errore, ma anche commenti inseriti nelle righe (i commenti sono preceduti dal #);
• Arancione: parole chiave

SCRIVERE ED ESEGUIRE UN PROGRAMMA

Per cominciare a scrivere un nuovo programma, è necessario aprire l'IDLE, come già accennato. Si possono mantenere aperte sia la finestra dell'interprete Python, sia quella dell'IDE. Una volta scritto il codice nell'IDE, bisogna salvare il file utilizzando il pulsante *File -> Salva come*, che consente di salvare un file che non è ancora stato rinominato, oppure *File -> Salva*, che viene utilizzato per i file già rinominati e che hanno subito delle modifiche. Inoltre, è possibile salvare una copia del file originale mantenendolo aperto (*File -> Salva copia come*). Una volta salvato il file, è possibile eseguire il programma (*Esegui -> Esegui modulo*) o effettuare un controllo della sintassi che rileva ed evidenzia eventuali errori nel codice (*Esegui -> Controlla modulo*). Se si decide di eseguire il programma senza aver effettuato il controllo, se vi sono errori, verrà visualizzato un messaggio di sintassi non valida e la posizione dell'errore verrà evidenziata in rosso nell'editor prima dell'esecuzione del programma nell'interprete Python.

Per aprire un file con estensione .py, non è sufficiente fare doppio clic sul file, sia che si voglia eseguire sia che si voglia semplicemente aprire. Per aprire un file di questo tipo, bisogna fare clic con il tasto destro del mouse e selezionare *Apri -> Modifica con l'IDLE*.

PRIME NOZIONI SULLA SINTASSI DI PYTHON

Ogni linguaggio di programmazione ha una propria sintassi; quella di Python è formata da keywords, operatori, punteggiatura e altri elementi che permettono di svolgere varie istruzioni.
Le regole sintattiche di un linguaggio stabiliscono come gli elementi devono essere "disposti" nelle righe al fine di creare delle istruzioni ben precise.

Python è un programma che non è rigido per quanto riguarda gli spazi, infatti è possibile inserire o meno gli spazi tra diversi elementi poiché il programma li riconoscerà a prescindere.

Le keywords di Python sono:

and	continue	finally	is	raise
as	def	for	lambda	return
assert	del	from	None	True
async	elif	global	nonlocal	try
await	else	if	not	while
break	except	import	or	with
class	False	in	pass	yield

In Python, oltre alle parole chiave, ci sono anche gli operatori matematici come +, -, , /, //, % e ** che possono essere utilizzati per eseguire operazioni matematiche tra numeri. Tuttavia, quando si utilizzano le stringhe di testo, questi operatori possono essere utilizzati solo per concatenare (+) o ripetere () le stringhe.

Non è possibile eseguire operazioni matematiche tra stringhe anche se il loro contenuto sembra numerico. Quando si utilizzano le stringhe di testo, come nella funzione *print*, è importante racchiudere la stringa tra apici (' ') o virgolette (" "), a seconda che sia necessario utilizzare le virgolette o gli apici nella stringa. Se la stringa di testo è su più righe, si possono utilizzare tripli apici o triple virgolette.
Inoltre, se si sta scrivendo una stringa di testo troppo lunga e si vuole andare a capo senza che l'output finale vada a capo, è possibile farlo inserendo un backslash (\ ) prima di andare a capo..

TIPI DI ERRORE

Vi sono due tipi di errore che possono presentarsi:

- **Errore di sintassi**: in questo caso verrà mostrato il messaggio SyntaxError: invalid syntax, se non è stata rispettata una regola di sintassi.
- **Errore durante l'esecuzione del programma:** quando la sintassi è corretta ma sorge un problema di esecuzione.
- **NameError:** si verifica quando si cerca di utilizzare una variabile o una funzione che non è stata definita o che non esiste nel contesto corrente.
- **TypeError**: si verifica quando si tenta di eseguire un'operazione non consentita su un determinato tipo di dato, ad esempio sommare una stringa con un intero.
- **IndexError:** si verifica quando si cerca di accedere ad un elemento di una lista o di una tupla utilizzando un indice che non esiste.
- **ValueError:** si verifica quando si passa un argomento di tipo corretto ma di valore inappropriato ad una funzione o ad un metodo.
- **KeyError:** si verifica quando si cerca di accedere ad un elemento di un dizionario che non esiste.
- **AttributeError:** si verifica quando si tenta di accedere ad un attributo o ad un metodo di un oggetto che non esiste.
- **ZeroDivisionError:** si verifica quando si cerca di dividere per zero.
- **KeyboardInterrupt:** si verifica quando l'utente interrompe l'esecuzione del programma premendo la combinazione di tasti Ctrl+C.

È possibile gestire gli errori attraverso le istruzioni *try/except* per evitare che il programma si interrompa in modo anomalo in caso di errore e per fornire un feedback più utile all'utente.

CODICI DI ESCAPE

I codici di escape (sequenze di escape o caratteri di escape) sono dei comandi preceduti dal backslash (\ ), posizionati all'interno di una stringa; non vengono visualizzati nell'output, ma danno comandi specifici. I codici di escape più comuni sono:

- **\n** ->manda a capo il testo;
- **\t** -> allinea alla tabulazione;
- **** -> manda a capo il codice;

FUNZIONI BUILT-IN

Le funzioni in Python sono delle porzioni di codice già predefinite che svolgono specifiche operazioni. Esse sono già presenti all'interno del programma, quindi sono chiamate built-in. Il nome di una funzione in Python è visualizzato in viola, mentre le stringhe di testo racchiuse tra virgolette o apici sono colorate in verde. Gli altri elementi della funzione, come ad esempio le variabili, sono visualizzati in **nero**. È importante ricordare di scrivere il nome della funzione in minuscolo, in quanto Python fa distinzione tra maiuscole e minuscole.

Ogni volta che si scrive il nome di una funzione e si apre la parentesi, viene mostrato un riquadro giallo contenente il cosiddetto call tip. Questo fornisce la sintassi della funzione e i vari parametri che possono essere inseriti, tutti separati da una virgola. È possibile modificare le impostazioni del call tip tramite il pulsante "Edit" e selezionando "Show Call Tip" se non è visibile.

Un esempio di funzione built-in è la funzione print, che consente di stampare una stringa di testo sullo schermo. I parametri fondamentali di questa funzione sono: "value", che rappresenta le stringhe o gli elementi da stampare, "sep=' '", che indica come separare i vari elementi, e "end='\n'", che indica come terminare la riga.

CALCOLI E OUTPUT DEI DATI

OPERATORI MATEMATICI E CALCOLI

Nei calcoli su Python i numeri sono detti operandi e le operazioni operatori matematici

Simbolo	Operazione	Descrizione	Esempi
+	**Addizione**	*Somma due numeri*	>>> 12 + 4 16
-	**Sottrazione**	*Sottrae due numeri*	>>> 45 -15 30
*	**Moltiplicazione**	*Moltiplica due numeri*	>>> 14 * 3 42
/	**Divisione**	*Divide due numeri e restituisce anche il resto*	>>> 56 / 3 18.6666666666666667
//	**Divisione intera**	*Divide e restituisce il numero intero, senza resto. (*)*	>>> 56 / 3 18
%	**Modulo (%)**	*Divide due numeri interi e restituisce il resto*	>>>16 % 3 1
**	**Esponente**	*Eleva a potenza*	>>>5 ** 2 25

N.B. con l'operazione divisione intera viene troncata la parte decimale non arrotondando se il numero è positivo come nell'es. ma arrotondando per difetto se il risultato è negativo.

FUNZIONE FORMAT

Per personalizzare l'aspetto di numeri in Python è possibile utilizzare la funzione format. Ad esempio, se volessimo formattare il risultato di 10 diviso 3 in modo che venga visualizzato con soltanto due cifre decimali, dovremmo scrivere format(10/3,'.2f'). In questo modo, l'output sarebbe 3.33.

La funzione format richiede due argomenti: il primo è il valore numerico che deve essere formattato e il secondo è una stringa di formattazione, indicata tra apici. Nel caso specifico, '.2f' indica a Python di visualizzare fino alla seconda cifra decimale, dove 'f' sta per float (cioè numero decimale).

Esistono anche altri stili di formattazione, come ad esempio '.%' che moltiplica il risultato per cento e quindi lo esprime come percentuale, oppure ',.2f' che, oltre a prendere due cifre decimali, separa le migliaia con una virgola.

ALTRE FUNZIONI MATEMATICHE

sum = somma elenco di elementi indicati tra parentesi	>>>sum((4, 5, 6)) 15
pow = elevamento a potenza	>>>pow(4, 2) 16
abs = valore assoluto	>>>abs(-8)8
max e min = calcolano max e min di un elenco di elementi tra parentesi	>>>max(1, 5, 7) 7
round = arrotonda un numero float alla cifra indicata come secondo elemento	>>>round(1.55, 1) 1.6

VARIABILI E TIPI DI DATI

VARIABILI E ISTRUZIONI DI ASSEGNAZIONE

Python è un linguaggio di programmazione che consente l'utilizzo di **variabili,** ovvero nomi che identificano un valore immagazzinato nella memoria del computer. Per utilizzare una variabile, è necessario prima inizializzarla, assegnandole un nome e un valore iniziale che poi rimarrà in memoria e che potrà essere modificato in seguito. La sintassi tipica per l'assegnazione di una variabile è la seguente: **nome_variabile = valore.** In questo caso, il simbolo "=" è chiamato operatore di assegnazione.

Le variabili sono estremamente utili perché consentono di memorizzare e manipolare dati all'interno del programma, in modo da semplificare la scrittura e la gestione del codice. Tuttavia, è importante ricordare che se si tenta di utilizzare una variabile che non è stata inizializzata, il programma restituirà un errore di tipo NameError, segnalando che il nome della variabile non è stato definito. Per evitare questo tipo di errore, è fondamentale assicurarsi di inizializzare sempre le variabili prima di utilizzarle.

```
>>> cognome_docente = 'Rossi'
>>> print('Il cognome del docente è:', cognome_docente)
Il cognome del docente è: Rossi
```

RIASSEGNAZIONE

Nel corso della scrittura del programma è possibile **riassegnare** un valore a una variabile tramite il processo di riassegnazione. Se per esempio abbiamo la variabile:
>>> a = 1
In seguito, potremo scrivere anche
>>> a = a +1
In questo modo il valore precedente viene eliminato dalla memoria in una sola volta.

TIPI DI DATI

Python adotta una **tipizzazione dinamica**, ciò significa che è il programma stesso a comprendere il tipo di variabile utilizzata, senza la necessità di una specifica dichiarazione da parte dell'operatore, a differenza di quanto avviene nella tipizzazione statica. I tipi di dati elementari in Python includono:

Tipo	Nome	Descrizione	Esempio
Intero	int	Numero intero, positivo o negativo	100, 0, -100
Reale	float	Numero a virgola mobile, positivo o negativo	100.9, 0.1, -100.5
Booleano	bool	Valore logico (vero o falso)	True, False
Stringhe	str	Stringa di testo alfanumerico	'Python' 'Web 2.0'

ESPRESSIONI CON TIPI DI DATI MISTI

Se si eseguono dei calcoli tra due operandi il tipo dei dati dipenderà dai due tipi di dati utilizzati:

- Calcoli tra due int danno come risultato un int
- Calcoli tra due float danno come risultato un float
- Calcoli tra un int e un float danno come risultato un float

FUNZIONI DI CONVERSIONE TRA TIPI

A volte è necessario convertire un valore da un tipo di dato in un altro per poter eseguire determinateoperazione, magari perché una funzione restituisce uno str quando invece ci serve un int o float.

FUNZIONE INT

La funzione **int** prende un valore e lo converte, se possibile, in numero intero.

```
>>> prezzo ='100'
>>> int(prezzo)
100
```

In questo modo abbiamo solo visualizzato il valore intero corrispondente ma non abbiamo convertito il tipo. Se invece operiamo in questo modo possiamo convertire il tipo di dato da str a int:

```
>>> prezzo = '100'
>>> prezzo_int = int(prezzo)
>>> prezzo_int
100
```

FUNZIONE FLOAT

La funzione **float** converte interi e stringhe (se possibile) in numeri a virgola mobile.

```
>>>   float(20)
20.0
```

FUNZIONE STR

Essa converte il valore del suo argomento in una stringa di testo. Questa funzione è molto utile quando vogliamo visualizzare risultati che siano composti sia da numeri che da stringhe (es. se vogliamo visualizzare un formato in valuta:

```
>>> profit = 100
>>> '€' + profit
```

Mostra un errore, in questo caso dobbiamo prima convertire il valore numerico in stringa.

```
>>> profit = 100
>>> profit_str = str(profit)
>>> '€'+profit_str
'€100'
```

LA FUNZIONE INPUT

La funzione **input** consente all'utente di immettere dei dati nella tastiera, che verranno poi utilizzati dal programma.

È molto frequente che venga utilizzata insieme alle variabili, in modo che ciò che viene digitato possa essere utilizzato in altre funzioni come variabile.

```
>>> variabile = input(prompt)
```

Dove prompt indica ciò che viene chiesto all'utente.

Il tipo restituito è sempre uno str anche se si tratta di un valore numerico, infatti se cercassimo di incrementare la variabile il programma ci restituirebbe un errore:

```
>>> eta = input('Quanti anni hai? ')
Quanti anni hai? 20
>>> eta = eta + 1
```

Quindi è necessario scrivere:

```
>>> eta =int(input('Quanti anni hai? ')
```

STRUTTURE DECISIONALI E LOGICA BOOLEANA

L'ISTRUZIONE IF

Fino ad ora abbiamo esaminato solo codici sequenziali, ovvero quei codici in cui le istruzioni vengono eseguite una dopo l'altra. Tuttavia, in alcune situazioni, il programma deve fare scelte diverse in base a determinate condizioni o valori di input. In questi casi, si utilizzano le **strutture decisionali**.

In Python, le strutture decisionali vengono implementate attraverso l'uso di istruzioni condizionali, come **l'if.** L'istruzione if consente di eseguire determinate istruzioni solo se una determinata condizione è vera. La sintassi dell'if è la seguente.

```
if consizione:
    'istruzione a'
    'istruzione b'
    ...
```

Subito dopo l'if è presente la condizione che se è vera allora si eseguono le istruzioni a, b ecc., se invece è falsa allora si salta il blocco di istruzioni.

ESPRESSIONI BOOLEANE

Le condizioni dell'istruzione if sono **espressioni booleane**, ovvero che possono restituire (True o False, N.B. la maiuscola è necessaria) a seconda che siano vere o false. Questi valori non sono stringhe.

Operatore	Descrizione	Esempi
==	Uguale a	>>> 3 == 3 True
!=	Diverso da	>>> 3 != 3 False
>	Maggiore di	>>> 7 > 5 True
>=	Maggiore o uguale a	>>> 8 >= 9 False
<	Minore di	>>> 8 < 10 True
<=	Minore o uguale a	>>> 5 <= 1 False

L'ISTRUZIONE IF-ELSE

Si analizza ora una struttura decisionale ad alternativa doppia, come segue

```
if condizione:
    istruzione a
    istruzione b
else:
    istruzione c
    istruzione d
```

Con tale struttura si procede in modo duale, se la condizione dopo l'if è vera si procede con le istruzioni a e b, se è falsa con le istruzioni dell'else, c e d.

Es.

```
x = int(input('Che ore sono?')
if x >= 23:
        print('Ricorda che domani hai lezione alle 8:45')
else:
        print("E' ancora presto vai tranquillo!")
```

OPERATORI LOGICI AND, OR, NOT

Espressione	Significato
x > y and a > b	x è maggiore di y E contemporaneamente a è maggiore di b?
x == y or v != z	x è uguale a y O v è diversa da z?
not (a >= b)	L'espressione a >= b è falsa?

N.B. Vi è inoltre l'operazione pass, che è un'operazione nulla utile per lasciare funzioni in sospeso, es:

```
if condizione1:
    pass
```

COSTRUTTI ITERATIVI

I CICLI

I cicli in Python consentono di eseguire ripetutamente lo stesso blocco di codice. Ci sono due tipi di cicli principali:

• I cicli condizionati, che si ripetono fino a quando una specifica condizione è vera o falsa.
• I cicli basati su contatore, che si ripetono un numero specifico di volte. In questo caso, il numero di ripetizioni viene stabilito tramite un contatore che viene aggiornato ad ogni ciclo.

IL CICLO WHILE

L'istruzione while permette di creare un ciclo controllato da una condizione ed è composta da

- La clausola while, con una condizione che assume i valori booleani (vero/falso) e termina con i duepunti
- Un'istruzione o un blocco di istruzioni che risultano indentate

```
while condizione:
    istruzione1
    istruzione2
    ...
```

Ciò s ando la condizione di partenza sarà vera, il blocco di istruzioni inserite verrà la condizione diventa falsa allora si esce dal ciclo while e si passa all'e uali altri comandi.

Es.

```
countdown = 5
while countdown >=1:
    print(countdown)
    countdown = countdown-1
print('Fine')
```

Risultato:

```
5
4
3
2
1
Fine
```

Se il ciclo non ha una fine facilmente definibile è possibile utilizzare non while ma **while True**, con il quale non è necessario ribattezzare la variabile prima di inserirla nel ciclo.
In while True è necessario, tuttavia, utilizzare la keyword if, la cui istuzione deve essere la keyword break (permette di uscire dal ciclo).
Es.

```
while True:
    input('Inserisci nome: ')
    input('Inserisci cognome: ')
    continua_inserimento = input('Vuoi continuare a inserire nomi? (s/n) ')
    if continua_inserimento == 'n':
        break
print('Sono stati inseriti tutti i nominativi')
```

IL CICLO FOR

Il ciclo for, invece, è un ciclo controllato da un contatore, in quanto le istruzioni vengono ripetute per un numero determinato di volte.
Esso è composto da:

- La clausola for con il nome di una variabile contatore, la parola in (ed eventualmente il range) e i due punti di chiusura.
- Un'istruzione o un blocco di istruzioni che risultano indentate

Alla variabile contatore viene assegnato il primo valore della lista o del range (a seconda che vi sia in o in range) e alle esecuzioni successive il valore della variabile viene incrementato, fino alla fine della lista o del range. Il numero di esecuzioni è quindi limitato.
Es.

```
for variabile in [val1,val2,val3]:
    istruzione 1
    istruzione 2
    ...
```

oppure

```
for variabile in range(start, stor, step):
    istruzione 1
    istruzione 2
    ...
```

LA FUNZIONE RANGE

Parliamo adesso della funzione **range**, la quale è utilizzata nel ciclo for. Si tratta di una funzione built-in di Python, ovvero una funzione inclusa nel linguaggio stesso. La sintassi di questa funzione prevede l'utilizzo di tre argomenti: **start, stop e step**. L'argomento start rappresenta il numero intero di partenza, mentre l'argomento stop indica il numero intero di arrivo. Infine, l'argomento step rappresenta gli eventuali intervalli tra un numero e l'altro mostrati dalla funzione, di default impostato a 1.

Nel caso in cui si ometta l'argomento start, la funzione inizierà a contare da zero. È importante notare che l'argomento stop è escluso dal conteggio, ovvero il numero di arrivo non è incluso nella sequenza di numeri restituita dalla funzione. Ad esempio, se utilizziamo range(1,5), verranno restituiti i numeri 1, 2, 3 e 4. Se invece vogliamo iniziare a contare da zero, possiamo utilizzare range(5). Se volessimo invece contare di due in due, possiamo utilizzare range(1,5,2), che restituirà i numeri 1 e 3. In alternativa, possiamo utilizzare range(5,1,-1) per contare all'indietro, partendo da 5 e diminuendo di uno alla volta, fino a raggiungere 1. Infine, se inseriamo un numero di tipo float all'interno della funzione range, otterremo un errore di tipo TypeError, che ci informerà che "un oggetto float non può essere interpretato come un intero".

CICLI NIDIFICATI

È possibile creare dei **cicli nidificati**, inserendo for e while all'interno di altri cicli.
Possiamo per esempio inserire un ciclo for in un altro ciclo for, oppure un ciclo for in un ciclo while, oppure, ancora, un ciclo while all'interno di un altro ciclo while.

```
for a in range(1,4):
    print('****', a)
    for b in range(3,0,-1):
        print('----', b)
print('Fine')
```

ISTRUZIONI BREAK E CONTINUE

All'interno dei cicli, è possibile utilizzare le istruzioni break e continue. L'istruzione **break** viene utilizzata per interrompere immediatamente l'esecuzione del ciclo e uscire da esso. D'altro canto, l'istruzione **continue** viene utilizzata per saltare l'iterazione corrente del ciclo e passare alla successiva, senza interrompere completamente l'esecuzione del ciclo. In questo modo, il programma continuerà ad eseguire le successive iterazioni del ciclo.

FUNZIONI

Oltre alle funzioni già presenti in Python è possibile andare a crearne di personalizzate, a cui si assegna un nome nel programma stesso. L'uso di funzioni personalizzate consente di costruire programmi modulari, i quali portano numerosi vantaggi: possibilità di riutilizzare il codice, maggiore semplicità del codice e miglioramento della fase di test.

DEFINIRE E CHIAMARE UNA FUNZIONE

Una funzione in Python si definisce così:

- Comando "def".
- Nome della funzione, che deve rispettare alcune regole (come nei nomi delle variabili) ossia non avere spazi, la prima lettera non può essere un numero e non si possono usare parole chiave di Python come return, while ecc. Inoltre, il nome dovrebbe rispettare il contenuto della funzione.
- Si mettono tra parentesi i parametri che indicano gli argomenti d'input che serviranno per le istruzioni della funzione, si mettono quindi i due punti.
- Si scrivono le istruzioni in dentatura (spostate a destra rispetto alla riga di def)

```
def nome_funzione (paramentri):
    istruzione1
    istruzione2
    ...
```

Vi possono essere però anche funzioni che non necessitano parametri, in tal caso si scrivono comunque le parentesi ma si lasciano vuote.

Per chiamare (far partire) una funzione è sufficiente digitare nella shell il nome della funzione e i parametri se necessari:

```
>>> nome_funzione(parametri)
```

ARGOMENTI DI UNA FUNZIONE

Vediamo ora come si usano i parametri di una funzione:

```
def sottrai (a,b):
    print ('Il risultato della sottrazione è:', a-b)
```

I parametri sono divisi tra loro da virgole e si usano nel definire una funzione, ossia quando a e b avranno dei valori (numerici nel nostro caso) allora saranno richiamati dalla funzione e diverranno argomenti della funzione stessa. I parametri sono quindi dei segna posti degli argomenti:

```
>>> a = 9
>>> b = 4
>>> sottrai (a, b)
Il risultato della sottrazione è: 5
```

Vi possono essere parametri obbligatori e opzionali, i primi sono necessari mentre i secondi se non vengono definiti hanno un valore predefinito da assumere:

```
def sottrai (a , b=4)
print (' il risultato della sottrazione è: ', a - b)
```

In questo caso se omettessimo di assegnare un valore a b, la funzione partirebbe lo stesso considerando b=4.

In Python i parametri vengono normalmente presi in ordine:
>>>sottrai(7, 4)
In tal caso a = 7 e b = 4

È possibile comunque scegliere quale parametro, assumerà quale valore; in tal caso gli argomenti sono detti argomenti denominati.
>>> sottrai(b = 7, a = 5)
-2

FUNZIONI PRODUTTIVE

Se in una funzione inseriamo l'istruzione return alla fine, allora il risultato della funzione rimane salvato in memoria e lo si può richiamare, in tal caso la funzione è detta produttiva:

```
def calcola(a, b, c):
    risultato = a * b / c
    return risultato
```

```
>>> calcola(3,4,5)
2.4
```

Se una funzione non è produttiva (es. tutte le precedenti a questo paragrafo) è detta funzione void, in quanto non mantiene il risultato.

VARIABILI LOCALI, VARIABILI GLOBALI E DOCSTRING

Le variabili locali sono variabili definite all'interno di una funzione e quindi utilizzabili solo dalle istruzioni all'interno della stessa. Le variabili locali invece sono utilizzabili da tutte le funzioni e istruzioni del programma.

```
def numeri(a, b):
    c = int(input('Inserisci il valore di c: '))
    valore1 = a**2
    valore2 = c + b / a
    return valore1 + valore2
```

```
>>> x = numeri(2,10)
Inserisci il valore di c: 15
>>> x
24.0
```

Ora mentre x è variabile globale, 'c' è invece locale. Se infatti provassimo a richiamare c si avrebbe un messaggio di errore.
È inoltre possibile inserire una stringa di documentazione detta docstring, ossia un commento che spieghi o dia informazioni sulla funzione. Tale docstring va inserita tra triple virgolette e subito dopo la definizione della funzione:

```
def funzione (parametri):
"""Questa è una funzione di prova"""
    istruzione1
    istruzione2
    ...
```

FUNZIONI CON CICLI

Le funzioni possono avere al loro interno anche cicli o strutture condizionali (if, elif, else). Non ci sono differenze di funzionamento rispetto alle funzioni già viste, vediamo un esempio per completezza:

```
def test_range (num):
    if num in range (1, 100):
        totale = 0
        for numero in range(1, num+1):
            totale = totale + numero
        return totale
    elif num > 100:
        print('Il numero deve essere compreso tra 1 e 100')
    else:
        print('Il numero deve essere maggiore di 0')
```

Dalla shell verifichiamo come nel caso della x il valore si sia salvato grazie al "return" (sesta riga), mentre nel caso della y no, poiché le altre funzioni sono void.

```
>>> test_range(12)
78
>>> x=test_range(12)
>>> x
78
>>> y=test_range(110)
Il numero deve essere compreso tra 1 e 100
>>> print(y)
None
```

STRINGHE E LISTE

SEQUENZE

Una sequenza è un oggetto che contiene diversi dati. Le forme più comuni e semplici di sequenze sono le stringhe e le liste. Esse presentano delle differenze, ma hanno anche delle affinità: per esempio, sia per le liste che per le stringhe possono essere utilizzate varie funzioni e operazioni.

STRINGHE

In Python, una stringa è una sequenza di caratteri alfanumerici. Per delimitare il principio e la fine di una stringa, si utilizzano gli apici singoli (') o le virgolette doppie ("). Una caratteristica delle stringhe è la loro immutabilità, il che significa che per modificare il contenuto di una stringa assegnata a una variabile, è necessario assegnare alla variabile una nuova stringa corretta oppure creare una nuova variabile contenente la stringa modificata.

OPERAZIONI SULLE STRINGHE

In Python è possibile effettuare delle operazioni con le stringe, alcune delle quali attraverso operatori matematici come + o *.

Operatore	Operazione
+	Concatena più stringhe senza lasciare spazi. Non può concatenare tipi di dati diversi. Può essere utilizzato nelle funzioni.
*	Crea una stringa ripetendo n volte la stringa inziale.
in	Restituisce True se la stringa è contenuta in un'altra, altrimenti False.
not in	Funziona in maniera esattamente contraria alla funzione in.
is	Restituisce True se una stringa è uguale all'altra, altrimenti False.
is not	Funziona in maniera esattamente contraria alla funzione is.

Esempi:

>>> testo = 'buon' + ' giorno' 'buongiorno'

>>> 'Python'*2 'PythonPython'

>>> x='Roma è la capitale' >>> 'capitale' in x True

LISTE

Una lista è una raccolta di diversi dati, ciascuno dei quali chiamato elemento. I vari elementi di una lista sono racchiusi tra parentesi quadre e divisi da virgole. Le liste possono contenere, tipi di dati diversi, quindi non devono necessariamente essere uniformi. Se volessimo visualizzare il contenuto di una lista possiamo utilizzare la funzione print(lista).

Esiste anche una funzione list, che permette di creare una lista in questo modo:

```
>>> testo=list('Milano')
>>> print(testo)
['M', 'i', 'l', 'a', 'n', 'o']
```

OPERAZIONI SULLE LISTE

Per le liste è possibile utilizzare alcuni degli operatori utilizzabili per le stringhe:

Operatore	Operazione
+	Unisce più liste, anche se si tratta di tipi di dati diversi.
*	Crea una lista composta dalla ripetizione della lista iniziale.
in	Restituisce True se l'elemento è contenuto nella lista, altrimenti False.
not in	Funziona in maniera esattamente contraria alla funzione in.

INDICIZZAZIONE

L'indicizzazione (indexing) è un modo per accedere a singoli elementi di stringhe o liste. Ogni elemento ha, infatti, una determinata posizione in una sequenza. È bene notare che se il conteggio parte da sinistra allora si avranno numeri positivi (a partire da 0), se parte da destra invece i numeri saranno negativi (a partire da -1).
L'indicizzazione avviene in questo modo:
>>>sequenza[indice]
Es.

```
>>> capitali = ['Roma','Parigi','Tokyo']
>>> capitali[2]
'Tokyo'
```

N. 1, quello di 'Tokyo' è 2.

SLICING

Si può selezionare una parte specifica di una stringa o di una lista usando lo slicing e gli indici, che vengono indicati come segue:

sequenza[indice_iniziale:indice_finale]

Va tenuto presente che l'indice iniziale (ovvero il primo carattere della stringa o l'elemento della lista) è incluso nella selezione, mentre l'indice finale è escluso. Se l'indice iniziale viene omesso, la selezione inizia dalla posizione 0, mentre se viene omesso l'indice finale, la selezione viene fatta fino alla fine della sequenza. È possibile anche specificare uno "step" per selezionare posizioni equidistanti, come nell'uso della funzione range. Nel caso in cui si usino indici negativi, anche lo step deve essere negativo. Se si vuole usare uno step negativo con indici positivi, invece, bisogna invertire l'ordine degli indici iniziale e finale.

FUNZIONI E METODI DELLE STRINGHE

Esistono in Python delle funzioni che operano sulle stringhe e che prevedono, nella loro sintassi, che almeno uno dei loro argomenti sia una stringa di testo o una variabile contenente una stringa di testo.

Funzione	Descrizione
len()	Restituisce la lunghezza di una stringa, contando i caratteri. È spesso usata nei cicli dentro alla funzione range.
min()	Restituisce il valore più piccolo, se si tratta di lettere la più piccola è la 'a'. Distingue tramaiuscole e minuscole
max()	Restituisce il valore più grande, se si statta di lettere la più grande è la 'z'. Distingue tra maiuscole e minuscole.

I metodi sono delle funzioni legate ad un preciso oggetto e seguono questa forma:stringa.metodo(argomenti).

Metodo	Descrizione
.upper()	Restituisce la stringa tutta in caratteri maiuscoli.
.lower()	Restituisce la stringa tutta in caratteri minuscoli.
.capitalize()	Restituisce la stringa con la prima lettera maiuscola e le altre minuscole.
.strip()	Restituisce la stringa dove gli spazi iniziali e finali sono stati rimossi.
.find(sub)	Cerca nella stringa la sottostringa specificata (sub) e restituisce l'indice della prima occorrenza trovata. Se la sottostringa non viene trovata viene restituito -1.
.replace(old,new)	Restituisce la stringa in cui tutte le occorrenze della sottostringa old sono state sostituite dalla new.
.startswith(prefix)	Restituisce True se viene trovato, all'inizio, il prefix, altrimenti viene restituito False.
.endswith(subfix)	Restituisce True se viene trovato, alla fine, il subfix, altrimenti viene restituito False.
.count(sub)	Restituisce il numero delle volte in cui compare nella stringa la sottostringa cercata.
.split(iterable)	Divide le stringhe ad ogni spazio (predefinito) e restituisce una lista di stringhe. È possibile inserire un separatore diverso come argomento. Può anche essere aggiunto come argomento anche maxsplit che indica il numero massimo di divisionida effettuare (quello predefinito è -1 che indica di separare fino alla fine della stringa).
.join(iterable)	Concatena gli elementi di una lista contenente solo stringhe e crea una stringa unica. Il metodo si applica al separatore (es. separatore_stringa = '/ ') mentre la lista è l'argomento tra parentesi. Es. separatore_stringa.join(lista)

FUNZIONI DELLE LISTE

Oltre alle funzioni che operano su stringhe, è possibile utilizzare delle funzioni built-in sulle liste:

Funzione	Descrizione
len()	Restituisce la lunghezza di una lista, cioè il numero di elementi.
min()	Restituisce l'elemento che inizia con il valore più piccolo. Nel caso di lista con elementi misti la funzione restituisce errore. Distingue tra maiuscole e minuscole.
max()	Restituisce l'elemento che inizia con il valore più grande. Nel caso di lista con elementi misti la funzione restituisce errore. Distingue tra maiuscole e minuscole.
list()	Converte un oggetto iterabile in una lista.
sorted()	Restituisce una lista ordinata in maniera crescente. Nel caso di lista con elementi misti la funzione restituisce errore.
sum()	Restituisce la somma di tutti gli elementi della lista, solo nel caso siano tutti numeri.

METODI DELLE LISTE

Allo stesso modo vi sono molti metodi che è possibile utilizzare sulle liste (alcune delle quali però non funzionano su liste con valori misti):

Metodo	Descrizione
.append(element)	Accoda nuovi elementi alla lista. È possibile utilizzarla anche nelle istruzioni dei cicli (ricordando di creare una lista vuota a cui verranno aggiunti dei valori. Vedi esempio).
.insert(index,element)	Inserisce l'elemento desiderato nella posizione specificata dal parametro index. N.B l'elemento precedente presente in quella posizione non viene eliminato, semplicemente si sposta.
.remove(element)	Cerca l'elemento richiesto ed elimina la prima occorrenza.
.pop([index])	Cerca l'elemento corrispondente all'indice inserito e lo elimina, se si omette l'indice viene eliminato l'ultimo elemento. Il metodo .pop restituisce l'elemento eliminato, che può essere riutilizzato in altre operazioni.
.extend(list2)	Accoda la lista indicata nell'argomento alla lista di partenza.

TUPLE E DIZIONARI

TUPLE

Una tupla è una sequenza di elementi racchiusi tra parentesi tonde e separati tra loro da una virgola (in realtà potrebbero funzionare anche senza parentesi, ma con l'uso di funzioni Python potrebbe non capire più dove inizia e dove finisce una tupla). Tutte le tipologie di elementi possono far parte di una tupla: numeri, stringhe, altre tuple ecc.

Esempio: numeri = (10, 20, 30)

Le tuple a prima vista potrebbero sembrare uguali alle liste, ma vi è un'importante differenza: le tuple sono immutabili, ciò le rende più sicure e più veloci. Una volta assegnati i valori agli elementi di una tupla, non è possibile modificarli. Ciò può essere utile quando si vogliono definire valori costanti o quando si vuole proteggere una sequenza di dati da eventuali modifiche accidentali.

È comunque possibile convertire una lista in tupla (con il metodo "tuple") e una tupla in lista (con il metodo "list").

Esempio: lista_numeri = [6, 24, 41, 90, 18, 37, 57, 5, 71] numeri2 = tuple(lista_numeri)

Comando	Significato dello slicing	Risultato
numeri2[4]	Selezione l'elemento della tupla con indice pari a 4.	18
numeri2[-6]	Seleziona il sesto elemento da destra.	90
numeri[2:5]	Seleziona gli elementi con indice da 2 (compreso) a 5 (escluso).	(41, 90, 18)
numeri2[:3]	Seleziona gli elementi dal primo a quello con indice 3 (escluso).	(6, 24, 41)
numeri2[:7:2]	Seleziona gli elementi dal primo a quello con indice 7 (escluso) di "2 in 2".	(6, 41, 18, 57)
numeri2[-5:-2]	Seleziona gli elementi da quello con indice -5 a quello con indice -2(escluso).	(18, 37, 57)

OPERATORI, FUNZIONI E METODI DELLE TUPLE

numeri = (1, 2, 3)
città = (Milano, Palermo, Viareggio)

Operatore	Operazione
+	Unisce due tuple in una nuova tupla.
*	Crea più copie di una tupla e le unisce.
in	Restituisce True.
not in	Seleziona gli elementi dal primo a quello con indice 3 (escluso).

Es. con "+" : nuova_tupla = numeri + città = (1, 2, 3, Milano, Palermo, Viareggio)
Es. con "*" : nuova_tupla = città*2 = (Milano, Palermo, Viareggio, Milano, Palermo, Viareggio)

Ecco invece le funzioni built-in utilizzabili con le tuple:

Funzione	Descrizione
len()	Restituisce il numero di elementi di una tupla.
tuple()	Converte un oggetto iterabile (lista) in tupla.
max()	Restituisce l'elemento più grande (se stringhe dal punto di vista alfabetico). Distingue tra maiuscole e minuscole.

min()	Restituisce l'elemento più piccolo (se stringhe dal punto di vista alfabetico). Distingue tra maiuscole e minuscole.
sorted()	Restituisce una nuova lista con gli elementi della tupla in ordine crescente.
sum()	Restituisce la somma degli elementi della tupla.

Si passa quindi a vedere i metodi utilizzabili con le tuple:

Metodo	Descrizione
.index(element)	Restituisce l'indice dell'elemento cercato, se ve ne sono più copie, prende il primo
.count(element)	Restituisce il numero di volte che un elemento compare in una tupla

DIZIONARI

Un dizionario è una struttura dati che contiene una raccolta di elementi. Ogni elemento di un dizionario è composto da una chiave, che deve essere immutabile e univoca all'interno del dizionario, e un valore, che può essere di qualsiasi tipo, mutabile o immutabile.

Per aggiungere un nuovo elemento al dizionario, o per sovrascrivere un elemento esistente, è possibile utilizzare la seguente sintassi: mio_dizionario[nuova_chiave] = nuovo_valore

Per accedere al valore di un elemento del dizionario, basta indicare la chiave corrispondente: valore_corrispondente = mio_dizionario[chiave]

Per eliminare un elemento del dizionario, o il dizionario intero, si può utilizzare il comando del: del mio_dizionario[chiave_da_eliminare] del mio_dizionario

Se si desidera accedere a più valori contemporaneamente, è possibile utilizzare un ciclo for per iterare su tutte le chiavi del dizionario e accedere ai valori corrispondenti: for chiave in mio_dizionario: **valore_corrispondente = mio_dizionario[chiave]**

OPERATORI, FUNZIONI E METODI DEI DIZIONARI

Con i dizionari, data la struttura dei propri elementi costituita da chiave e valore, non è possibile usare gli operatori tradizionali (somma, moltiplicazione ecc.). Si possono invece usare degli operatori logici:

Operatore	Operazione
in	Restituisce True se una chiave è inclusa in un dizionario, altrimenti restituisce False.
not in	Restituisce True se una chiave non è inclusa in un dizionario, altrimenti restituisceFalse.
is	Restituisce True se due dizionari sono identici, altrimenti restituisce False.
is not	Restituisce True se due dizionari non sono identici, altrimenti restituisce False.

Si passa quindi a vedere le funzioni built-in utilizzabili con i dizionari:

Funzione	Descrizione
len()	Restituisce il numero di elementi (coppie chiave-valore) di un dizionario.
dict()	Crea un dizionario vuoto, a meno che come argomento della funzione si abbiano degli oggetti iterabili.
max()	Restituisce la chiave più grande di un dizionario (se le chiavi sono stringhe, allora restituisce la più grande dal punto di vista alfabetico). Distingue tra maiuscole e minuscole
min()	Restituisce la chiave più piccola di un dizionario (se le chiavi sono stringhe, allora restituisce la più piccola dal punto di vista alfabetico). Distingue tra maiuscole e minuscole
sorted()	Restituisce una lista con le chiavi del dizionario ordinate in maniera crescente (se le chiavi sono stringhe saranno messe in ordine alfabetico).

Vediamo ora i metodi dei dizionari:

Metodo	Descrizione
clear()	Rimuove tutti gli elementi di un dizionario, ma non elimina il dizionario.
.get(nome_chiave,[valore_default])	Restituisce il valore corrispondente alla chiave (indicata in nome_chiave), se la chiave non è presente nel dizionario restituisce None come valore predefinito o, se inserito, restituisce valore_default, che è quindi opzionale.
.pop(nome_chiave,[valore_default])	Rimuove la chiave (indicata in nome_chiave) e il relativo valore associato, inoltre restituisce il suddetto valore. Se la chiave non è presente nel dizionario restituisce KeyError come valore. predefinito o, se inserito, restituisce valore_default, che è quindi opzionale.
.popitem()	Rimuove l'ultima coppia chiave-elemento aggiunti al dizionario e restituisce i suddetti come tupla.
.update(nome_dizionario2)	Aggiunge le chiavi e i valori di un secondo dizionario (nome_dizionario2) a un dizionario esistente.
.items()	Crea un oggetto del tipo dict_items che contiene una lista dituple (ognuna di esse con la coppia chiave-valore.
.keys()	Crea un oggetto del tipo dict_keys che contiene una lista contutte le chiavi del dizionario.
.values()	Crea un oggetto del tipo dict_values che contiene una lista contutti i valori del dizionario.

ACCESSO AI FILE E GESTIONE DI ERRORI

ACCEDERE AI FILE

È possibile, all'interno di Python, aprire (o creare) dei file di testo e modificarli.Per accedere ad un file esistono due metodi:

- Accesso di tipo sequenziale (su cui ci focalizzeremo): si accede al file e lo si legge dall'inizio alla fine.
- Accesso di tipo diretto: si accede direttamente alla parte che ci interessa, è un processo più veloce.

Per aprire un file dobbiamo innanzitutto creare una variabile cui associare il cosiddetto 'oggetto file',scrivendo:
variabile = open(file,mode)
L'argomento file è obbligatorio e deve comprendere il percorso e il nome del file (se il percorso è omesso Python assume che la sua posizione sia la stessa del programma). Se nel percorso sono presenti dei '\' allora è necessario scrivere il percorso preceduto da una r, in modo che il '\' non venga considerato come carattere di escape:
L'argomento mode, invece, è opzionale e stabilisce in che modo aprire il file (sola lettura, scrittura ecc.), di default è impostato su sola lettura.

```
variabile = open(r'D:\Example\file.txt')
```

I modi in cui è possibile visualizzare il file sono:

Modalità	Descrizione
'r'	Apre il file in modalità sola lettura, quindi non è possibile scrivere o modificare nulla. All'apertura il programma si posiziona all'inizio della prima riga (posizione 0). Se il file non esiste viene restituito errore.
'w'	Apre il file in modalità scrittura. Se il file esiste già i dati preesistenti vengono eliminati, altrimenti viene creato un nuovo file.
'a'	Apre il file in modalità append (aggiunta), ciò vuol dire che è possibile scrivere alla fine dei dati già esistenti, poiché Python si posiziona alla fine del testo preesistente. Se il file non esiste viene creato.
'r+'	Apre il file in modalità lettura e scrittura. Se il file esiste già i dati non vengono eliminati, se non esiste viene restituito un errore.
'w+'	Apre il file in modalità lettura e scrittura. Se il file esiste già i dati vengono eliminati, se non esiste viene creato il nuovo file.
'a+'	Apre il file in modalità aggiunta e lettura. Se il file esiste già Python si posiziona dopo il testo esistente altrimenti il nuovo file viene creato.

Riscrivi: Per poter scrivere o leggere da un file, è necessario utilizzare dei metodi specifici, e alla fine della modifica bisogna chiudere il file con il metodo close() invocato sulla variabile che lo rappresenta.

Inoltre, esistono anche dei cosiddetti attributi del file, che possono essere accessibili tramite il nome della variabile del file, e che ci forniscono informazioni sul file stesso, ad esempio:

• **variabile.name**: restituisce il percorso e il nome del file.

• **variabile.mode** : restituisce la modalità di apertura del file.

• **variabile.closed** : restituisce True o False a seconda che il file sia chiuso o aperto.

A differenza dei metodi, per gli attributi non sono necessarie le parentesi e non richiedono argomenti.

METODI PER LEGGERE I DATI

I metodi che si utilizzano per leggere i dati e posizionarsi all'interno del testo sono molti. I più utilizzati sono:

Metodo	Descrizione
.read(size)	Legge il contenuto del file dalla posizione corrente fino alla fine. Se inseriamo l'argomento size(opzionale), Python legge il numero di byte impostato. Es. se size=10 il programma legge dalla posizione corrente fino a 10 battute in avanti.
.readline(size)	Legge il contenuto di una riga di testo. Anche qui size è opzionale.
.readlines(size)	Legge il contenuto di tutte le righe di testo fino alla fine del file e restituisce una lista che ha per elementi le singole righe. L'argomento size è opzionale.
.seek(offset, [whence])	Permette di spostare il puntatore alla posizione desiderata, specificando di quanti byteci si deve muovere. L'argomento offset indica il numero di byte, mentre l'argomento whence indica la posizione da cui partire: 0 = inizio file, 1 = posizione corrente, 2 = fine file. (Es. se vogliamo spostarci alla fine del file scriveremo file.seek(0,2).
.tell	Restituisce la posizione nel testo in termini di byte a partire dall'inizio. (tiene conto degli spazi e dei codici di escape).

METODI PER SCRIVERE SU UN FILE

Esistono anche dei metodi che permettono di scrivere su un file aperto in modalità scrittura o aggiunta:

Metodo	Descrizione
.writable()	Restituisce True se è possibile scrivere sul file, altrimenti False.
.write(string)	Scrive nel file il contenuto inserito come argomento string. Alla fine, Pyhon restituisce il numero di caratteri scritti e il puntatore nel file si posiziona alla fine del testo. Bisogna inserire nell'argomento string anche eventuali codici di escape.
.writelines(lines)	Scrive nel file la sequenza di righe specificate nell'argomento lines. Bisogna inserireanche i codici di escape.

TRY...EXCEPT

Qualora si prevede che una determinata istruzione possa dare un errore è possibile utilizzare le funzioni **try ed except** per evitare il messaggio di errore. Questo processo è detto di exception handling.

Es. vogliamo dividere due numeri ma il secondo è zero, oppure i numeri inseriti sono di tipo stringa; per evitare il messaggio ZeroDivisionError oppure ValueError possiamo scrivere:

Come si vede try ed except devono essere seguite da ' : ' e devono esserescritte con la stessa indentazione. Le istruzioni devono essere anche loro indentate, sullo stesso livello.

DEBUGGING

Il **debugging** in Python è il processo di individuare e risolvere gli errori (bug) in un programma. In pratica, il debugging è il processo di identificazione e correzione degli errori di codice che impediscono al programma di funzionare come previsto.

Il debugging in Python può essere fatto attraverso diverse tecniche, tra cui l'utilizzo di stampa a schermo, il controllo dei valori di una variabile in un dato momento del programma, l'utilizzo del modulo di logging, l'utilizzo di strumenti di debugging come PyCharm o Visual Studio Code.

Una delle tecniche di debugging più utilizzate in Python è il **breakpoint,** un punto di interruzione nel codice in cui il programma si arresta, consentendo all'utente di esaminare lo stato del programma e di identificare eventuali errori. Durante la modalità di debug, l'utente può interagire con il programma e modificare le variabili al fine di verificare il corretto funzionamento del programma.

In sintesi, il debugging è un'attività importante per garantire che il codice funzioni correttamente e che il programma produca i risultati attesi. Consiste nel trovare e risolvere gli errori nel codice attraverso l'uso di tecniche specifiche, come i breakpoint e il controllo dei valori delle variabili, al fine di garantire il corretto funzionamento del programma.

LE LIBRERIE DI PYTHON

LA LIBRERIA STANDARD DI PYTHON

La **libreria standard** di Python contiene oltre 180 moduli, più o meno complessi, analizziamo i più comuni *(N.B durante le lezioni sono stati utilizzate solo alcune funzioni dei primi 3):*

IL MODULO MATH

Esso permette di lavorare con numeri reali e le sue funzioni più usate sono:

Funzioni	Descrizione	Esempio
math.ceil(x)	Restituisce l'intero arrotondamento per eccesso di x	4.7 → 5
math.floor(x)	Restituisce l'intero arrotondamento per difetto di x	4.7 → 4
math.sqrt(x)	Restituisce la radice quadrata di x	6.25 → 2.5
math.pi	Esprime il Pi greco con approssimazione secondo la potenza del pc	3.14...
math.exp(x)	Restituisce e elevato alla potenza x (e = numero di Nepero)	
math.factorial	Restituisce il fattoriale del numero intero positivo x	4! → 4*3*2*1
math.gcd(a, b)	Restituisce il massimo comun divisore tra a e b	Tra 36 e 84 → 12
math.hypot(x, y)	Applica il teorema di Pitagora per calcolare l'ipotenusa tra x e y	rad(4*2+3*2) → 5
math.isfinite(x)	Restituisce True se x è diverso da 0/n e da 0/0, altrimenti dà False	
math.isnan(x)	Restituisce True se x è uguale a 0/0, altrimenti dà False	
math.log(x,[baseA])	Restituisce il logaritmo in base A di x, se baseA è omessa calcola il logaritmo in base 10	
math.logq (x)	Calcola il logaritmo in base q di x	
math.pow(x, y)	Eleva x alla y	3**4 → 81

IL MODULO RANDOM

Permette di utilizzare funzioni aleatorie per calcoli di probabilità:

Funzioni	Descrizione
random.random()	Restituisce un numero decimale compreso tra 0 (incluso) e 1 (escluso)
random.choice(sequenza)	Restituisce un elemento a caso tra quelli indicati nella sequenza (che può essere una lista, una tupla, una stringa ecc.)
random.randrange(stop) random.randrange(start, stop,[step])	Restituisce un numero intero scelto tra start (incluso, e se omesso uguale a 0) e stop andando di step in step (se omesso è uguale a 1); ilfunzionamento è identico alla funzione range.
random.randint(min, max)	Restituisce un numero intero compreso tra min e max (entrambi inclusi)
random.seed(a = None)	Inizializza il generatore di numeri casuali con il numero intero a dato; se omesso o None utilizza il valore dell'orologio interno del computer nel momento dell'esecuzione. Si usa se si vuole ottenere sempre la stessa sequenza di valori casuali.
random.shuffle(x)	Permuta casualmente l'ordine degli elementi della sequenza x.
random.sample(population, k)	Restituisce una lista di k elementi dalla lista population (senza reimmessa).
random.sample(population, weights = None, k = 1)	Estrae una lista di k elementi dalla sequenza population facendo campionamento (con reimmessa). È possibile rendere gli elementi equiprobabili impostando a weights un valore uguale alla lunghezza di population, il quale contiene le diverse probabilità degli elementi.
random.gauss(mu, sigma)	Genera un numero casuale da una variabile casuale normale di media mu e di scarto quadratico medio sigma.

IL MODULO TURTLE

Permette un uso grafico molto semplice:

Tartaruga: metodi	Descrizione
bob = turtle.Turtle()	Funzione per costruire un nuovo oggetto tartaruga di nome bob.
bob.forward(distanza)	La tartaruga bob si muove in avanti di distanza pixel.
bob.backward(distanza)	La tartaruga bob si muove indietro di distanza pixel.
bob.goto(x, y)	La tartaruga bob va alla posizione x = ascissa, y = ordinata.
bob.left(gradi)	La tartaruga bob ruota in senso antiorario di quanto indicato in gradi.
bob.right(gradi)	La tartaruga bob ruota in senso orario di quanto indicato in gradi.
bob.setheading(gradi)	La tartaruga bob si orienta rispetto alla posizione originale di tanto quanto indicato in gradi.

bob.penup()	Solleva la penna della tartaruga bob così che non scriva più.
bob.pendown()	Abbassa la penna della tartaruga bob così che scriva.
bob.begin_fill()	La tartaruga bob inizia a colora l'interno delle figure che disegna.
bob.end_fill()	La tartaruga bob smette di colorare l'interno delle figure che disegna.
bob.pencolor(colore)	Imposta il colore della penna della tartaruga bob.
bob.width(dimensione)	Imposta la larghezza della penna della tartaruga bob.
bob.fillcolor(colore)	Imposta il colore che la tartaruga bob utilizza per colorare l'interno delle figure che disegna.
bob.shape('turtle')	Cambia l'aspetto della tartaruga bob (es. con 'turtle' si avrà l'aspetto di una tartaruga).
bob.position()	Indica l'attuale posizione della tartaruga bob, i valori delle coordinate vengono restituiti in una tupla (x, y).
bob.heading()	Indica l'orientamento dei gradi della tartaruga bob.
canvas = turtle.Screen()	Costruisce un nuovo oggetto di nome canvas.
canvas.bgcolor(colore)	Imposta il colore della finestra background.
canvas.addshape("ninja.gif")	Aggiunge alla finestra l'immagine come nuova forma per il cursore.

IL MODULO OS E OS.PATH

Permettono di interagire con la gestione file e cartelle del sistema operativo:

Funzioni	Descrizione
os.listdir([path])	Restituisce la lista dei nomi dei file e delle directory presenti nella cartella identificata dal percorso path.
os.getcwd()	Restituisce una stringa che rappresenta il percorso dell'attuale working directory directory.
os.chdir(path)	Fa diventare il percorso path la nuova working directory.
os.rename(oldName, newName)	Rinomina il file o la directory.
os.remove(file)	Cancella il file.
os.rmdir(path)	Cancella la directory se vuota.
os.path.join(path, filename)	Restituisce una stringa concatenando path e filename, secondo la sintassi del computer.
os.path.isfile(path)	Restituisce True se il percorso path termina con il nome di un file.
os.path.isdir(path)	Restituisce True se il percorso path termina con una directory, altrimenti False.

IL MODULO DATETIME

Fornisce dati speciali connessi alle date:

Principali tipi di dati	Descrizione
x = datetime.date(year, month, day)	Crea un valore di tipo date con attributi: year, month, day.
y = datetime.time(hour, minute, second)	Crea un valore di tipo time con attributi: hour, minute, second.
z = datetime.timedelta([days,[seconds,[minutes,[hours,[weeks]]]]])	Crea un valore durata duration ottenuto dalla differenza tra due valori ditipo date o time.
x.weekday()	Restituisce il giorno della settimana come numero intero 0 - 6, dove illunedì vale 0 e la domenica vale 6.
x.isoformat()	Restituisce una stringa che rappresentala data in formato: YYYY - MM -DD.

CLASSI, ATTRIBUTI E METODI

IL LINGUAGGIO A OGGETTI

In Python, tutto è un oggetto e, come nella vita reale, ogni oggetto può essere usato in uno o più modi, a seconda del tipo di oggetto che stiamo utilizzando. Ci sono due tipi di azioni che possiamo svolgere con un oggetto: modificare le sue caratteristiche attraverso gli **attributi,** o interagire con esso e con altri oggetti attraverso i **metodi.**

La definizione di classe è un blocco di codice che contiene sia gli attributi che i metodi relativi ad un oggetto. È importante rendere il codice il meno complicato possibile, seguendo il principio del Don't Repeat Yourself (DRY), in modo da poterlo riutilizzare facilmente.

L'approccio alla programmazione orientata agli oggetti (OOP) si basa su tre principi fondamentali:

- **Incapsulamento**: questo meccanismo non rende visibili agli altri oggetti le caratteristiche di una classe. Questa caratteristica presenta vari vantaggi, tra cui la facilità di condivisione con altri programmatori, la semplicità e la velocità nel testare il funzionamento, la sicurezza del codice e l'aumento della produttività.

- **Ereditarietà:** questo meccanismo permette di costruire nuove classi a partire da classi preesistenti, ereditando attributi e metodi. Si parla di classi figlie/child e classi base/parent.

- **Polimorfismo**: questo meccanismo permette di dare lo stesso nome a metodi che agiscono su oggetti diversi ma alla stessa maniera.

LE CLASSI E LE ISTANZE

In Python, **una classe** è essenzialmente una famiglia o una collezione di oggetti di un certo tipo, che possono essere incorporati in Python o in librerie esterne, oppure possono essere creati dall'utente. **Un'istanza**, invece, è un singolo oggetto che appartiene a una classe. Utilizzando l'esempio di una classe di automobili, le istanze sarebbero i singoli modelli di auto.

Ogni oggetto deve avere attributi e metodi specifici. Per esempio, la classe dell'automobile avrebbe attributi come la potenza del motore, il tipo di trasmissione, ecc. e metodi come l'avvio, l'arresto, il cambio di marcia, ecc. L'insieme dei metodi di una classe si chiama interfaccia.

In Python esistono tipi di dati elementari come stringhe, interi, float e booleani, oltre a strutture di dati più complesse come tuple, liste e dizionari, che sono chiamati letterali. A differenza degli oggetti veri e propri, i letterali vengono creati utilizzando una sintassi dedicata, più semplice e riconoscibile, che permette a Python di identificare i vari letterali (ad esempio, le stringhe sono identificate da virgolette o apostrofi, gli interi da cifre da 0 a 9, e così via).

Come abbiamo già detto, è possibile cambiare il tipo di dati di una variabile utilizzando funzioni come str, int e float. In realtà, però, la conversione non modifica l'oggetto originale, ma ne crea uno nuovo sulla base del primo, riassegnando il nome e il tipo desiderato. In Python, infatti, il tipo è strettamente legato all'oggetto e non può essere modificato.

Esistono già classi nella libreria standard di Python e in librerie di terze parti. Esistono precise regole di denominazione del codice che consentono agli utenti di identificare quando stanno utilizzando o creando una classe. Una di queste regole consiste nell'iniziare il nome della classe con una lettera maiuscola.

Utilizzando la funzione type, è possibile verificare a quale classe appartiene un elemento, mentre per vedere il contenuto del modulo è necessario utilizzare la funzione dir(). Ad esempio, dir(turtle) visualizza l'intero contenuto del modulo turtle.

Per visualizzare solo gli attributi e i metodi, la funzione dir() deve essere usata con il nome della classe o dell'istanza. Ad esempio, dir(turtle.Turtle()). Infine, per ottenere una spiegazione più dettagliata, si possono usare sia la funzione di aiuto che il cosiddetto call tip. Il suggerimento di chiamata viene visualizzato quando il nome di un oggetto è seguito da un punto o quando la parentesi è aperta dopo un metodo.

BEST PRACTICE NELLA CREAZIONE DELLE ISTANZE

Per evitare che altri programmatori possano modificare il contenuto del codice è possibile rendere alcuni attributi costanti. Per far ciò è possibile utilizzare la funzionalità di tipo get. Invece, se si vuole rendere l'attributo modificabile bisogna utilizzare la funzionalità set.

```
def getValuta(self):
    return self.valuta
```

```
def setInizializz(self, newSeed):
    self.inizializ = newSeed
```

EREDITARIETÀ

L'ereditarietà si basa sul concetto di gerarchia, che in Python è particolarmente da creare e utilizzare. Basta indicare nella sottoclasse(child) il nome della classe base (parent).

```
class SuperClass():
    pass
class SubClass(SuperClass):
    pass
```

È anche possibile inserire più sopraclassi all'interno di una sottoclasse.

OVVERRIDING DI METODI

Si definisce overriding il meccanismo secondo il quale il quale Python, a parità di nome del metodo, privilegia l'esecuzione, nella sottoclasse, del metodo nella sottoclasse stessa. Questo perché si presuppone che il livello di specializzazione dei livelli più bassi sia maggiore.

```
class Dipendente():
    ore = 8
    retrOraria = 15
    def stipendio(self):
        print('La paga è €'+self.ore*self.stipendio*20)

class Parttime(Dipendente):
    def stipendio(self):
        print('Dato che sei part time, la paga è di €'+self.ore*self.stipendio*20*0.5)
```

Se noi cercassimo lo stipendio di un lavoratore part time, utilizzando la classe Parttime otterremmo l'esecuzione del secondo modulo dal nome stipendio.

OVERLOADING DI METODI

Il concetto di **overloading** di metodi si verifica quando gli operatori assumono comportamenti distinti a seconda del tipo di dati che manipolano. Questo comportamento è realizzato tramite la definizione di metodi specifici (ad esempio "add") per gestire gli operatori in modo differente. Questa funzionalità è già stata incontrata con gli operatori + e * che hanno un comportamento differente a seconda che operino su stringhe o numeri interi/numeri decimali.

POLIMORFISMO

Il **polimorfismo** è la capacità di un programma di utilizzare una stessa funzione o metodo in modi diversi a seconda del tipo di dato su cui viene applicata. In altre parole, la stessa funzione può comportarsi in modo diverso in base al tipo di dato su cui viene eseguita. Questa caratteristica consente di scrivere codice più flessibile e riutilizzabile in diversi contesti.

Per dubbi o suggerimenti sulla dispensa:

DOMENICO DESTITO

+39 3511013686

@destitodomenico

CHIARA TUA

+39 3479789059

@chiara_tua

Per info sulla nostra Area Didattica:

GIOVANNI BARBARO

+39 3277175240

@gianni_barbaro2

CARLOTTA CAROMANI

+39 3703723764

@carlottacaromani